

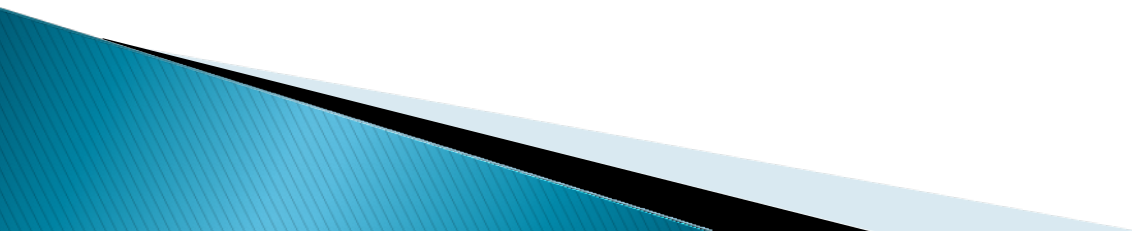
CSSE 220 Day 17

File I/O, Exceptions

Check out *FilesAndExceptions* from SVN

Questions?

File I/O: Key Pieces

- ▶ Input: **FileReader** and **Scanner**
 - ▶ Output: **PrintWriter** and **println**
 - ▶ Be kind to your OS: **close()** all files
 - ▶ Letting users choose: **JFileChooser** and **File**
 - ▶ Expect the unexpected: **Exception** handling
 - ▶ Refer to examples when you need to...
- 

Exceptions

- ▶ Used to signal that something went wrong:
 - `throw new EOFException("Uneven number of ints");`
- ▶ Can be **caught** by **exception handler**
 - Recovers from error
 - Or exits gracefully

A Checkered Past

- ▶ Java has two sorts of exceptions
- ▶ **Checked exceptions**: compiler makes sure that calling code doesn't ignore the problem if it occurs.
 - Used for **expected** problems
- ▶ **Unchecked exceptions**: compiler lets us ignore these if we want
 - Used for **fatal** or **avoidable** problems
 - Are subclasses of RuntimeException or Error

Hierarchy of Exception Classes

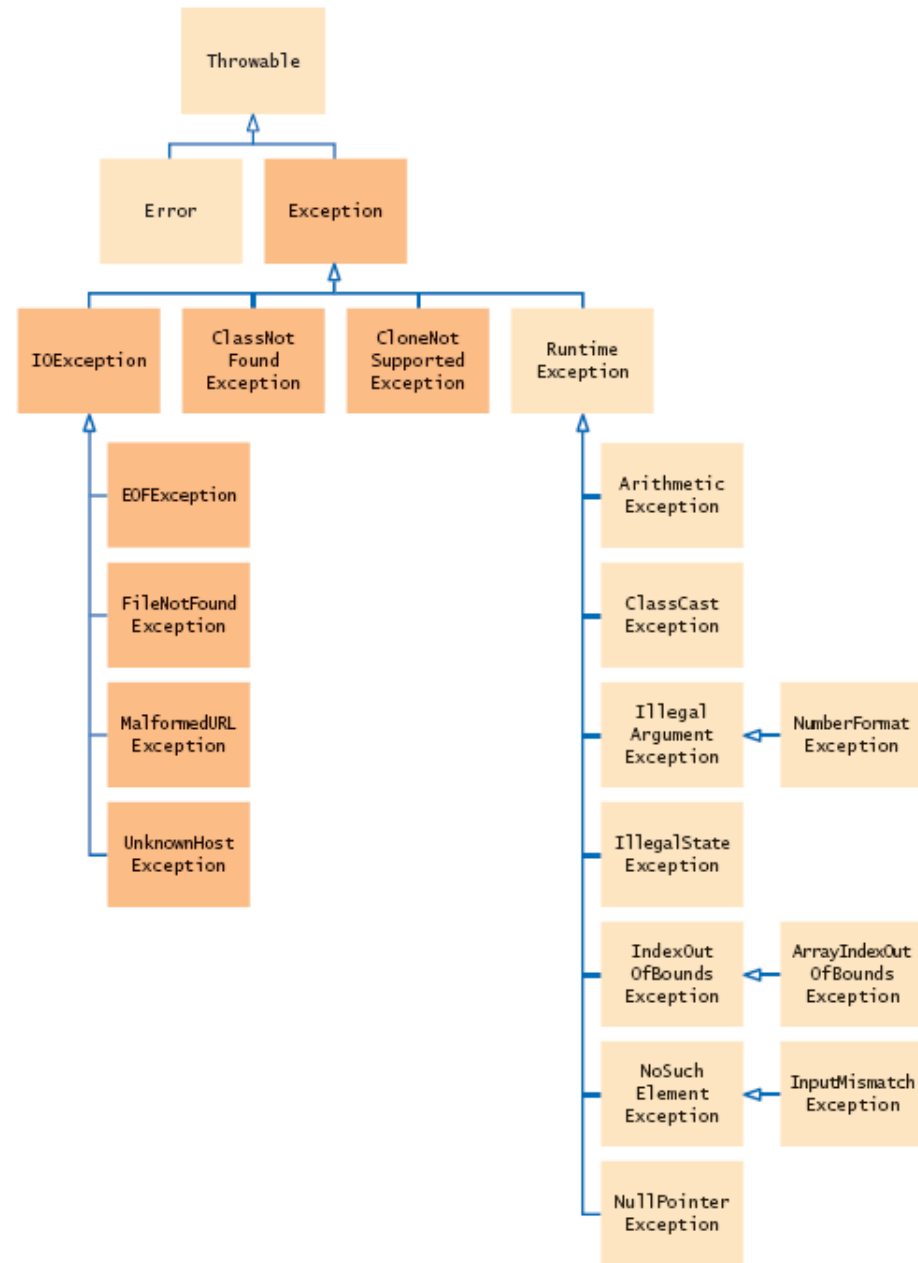


Figure 1 The Hierarchy of Exception Classes

A Tale of Two (and a half) Choices

- ▶ Dealing with checked exceptions
 - Can **propagate** the exception
 - Just declare that our method will pass any exceptions along
 - **public void loadGameState() throws IOException**
 - Used when our code isn't able to rectify the problem
 - Can **handle** the exception
 - Used when our code can rectify the problem
 - Can **do both**
 - Do what we can to handle the exception, and then throw the same (or a different) exception

Handling Exceptions

- ▶ Use try-catch statement:

- `try {`
 // potentially “exceptional” code
`} catch (ExceptionType var) {`
 // handle exception
`}`

Can repeat this part for as many different exception types as you need.

- ▶ Related, try-finally for clean up:

- `try {`
 // code the requires “clean up”
`} finally {`
 // runs even if exception occurred
`}`

BallWorlds Work Time

»» Ask questions if you're stuck!